LPI exam 301 prep, Topic 303: Configuration

Senior Level Linux Professional (LPIC-3)

Skill Level: Intermediate

Sean A. Walberg (sean@ertw.com)
Senior Network Engineer

04 Mar 2008

In this tutorial, Sean Walberg helps you prepare to take the Linux Professional Institute Senior Level Linux Professional (LPIC-3) exam. In this third in a series of six tutorials, Sean walks you through configuring a Lightweight Directory Access Protocol (LDAP) server, including access control, security, and performance. By the end of this tutorial, you'll know about LDAP server configuration.

Section 1. Before you start

Learn what these tutorials can teach you and how you can get the most from them.

About this series

The Linux Professional Institute (LPI) certifies Linux® system administrators at three levels: *junior level* (also called "certification level 1"), *advanced level* (also called "certification level 2"), and *senior level* (also called "certification level 3"). To attain certification level 1, you must pass exams 101 and 102. To attain certification level 2, you must pass exams 201 and 202. To attain certification level 3, you must have an active advanced-level certification and pass exam 301 ("core"). You may also need to pass additional specialty exams at the senior level.

developerWorks offers tutorials to help you prepare for the five junior, advanced, and senior certification exams. Each exam covers several topics, and each topic has a corresponding self-study tutorial on developerWorks. Table 1 lists the six topics and

corresponding developerWorks tutorials for LPI exam 301.

Table 1. LPI exam 301: Tutorials and topics

LPI exam 301 topic	developerWorks tutorial	Tutorial summary
Topic 301	LPI exam 301 prep: Concepts, architecture, and design	Learn about LDAP concepts and architecture, how to design and implement an LDAP directory, and about schemas.
Topic 302	LPI exam 301 prep: Installation and development	Learn how to install, configure, and use the OpenLDAP software.
Topic 303	LPI exam 301 prep: Configuration	(This tutorial) Lean how to configure the OpenLDAP software in detail. See the detailed objectives.
Topic 304	LPI exam 301 prep: Usage	Coming soon.
Topic 305	LPI exam 301 prep: Integration and migration	Coming soon.
Topic 306	LPI exam 301 prep: Capacity planning	Coming soon.

To pass exam 301 (and attain certification level 3), the following should be true:

- You should have several years of experience with installing and maintaining Linux on a number of computers for various purposes.
- You should have integration experience with diverse technologies and operating systems.
- You should have professional experience as, or training to be, an enterprise-level Linux professional (including having experience as a part of another role).
- You should know advanced and enterprise levels of Linux administration including installation, management, security, troubleshooting, and maintenance.
- You should be able to use open source tools to measure capacity planning and troubleshoot resource problems.
- You should have professional experience using LDAP to integrate with UNIX® services and Microsoft® Windows® services, including Samba, Pluggable Authentication Modules (PAM), e-mail, and Active Directory.
- You should be able to plan, architect, design, build, and implement a full

environment using Samba and LDAP, as well as measure the capacity planning and security of the services.

- You should be able create scripts in Bash or Perl or have knowledge of at least one system programming language (such as C).

The Linux Professional Institute doesn't endorse any third-party exam preparation material or techniques in particular.

About this tutorial

Welcome to "Configuration," the third of six tutorials designed to prepare you for LPI exam 301. In this tutorial, you learn about LDAP server configuration, including access controls, security, replication, and database performance.

This tutorial is organized according to the LPI objectives for this topic. Very roughly, expect more questions on the exam for objectives with higher weights, as shown in Table 2.

Objectives

Table 2 lists the detailed objectives for this tutorial.

Table 2. Configuration: Exam objectives covered in this tutorial

LPI exam objective	Objective weight	Objective summary
303.2 Access control lists in OpenLDAP	2	Plan and implement access control lists.
303.3 LDAP replication	5	Set up OpenLDAP to replicate data between multiple servers.
303.4 Securing the directory	4	Configure encrypted access to the LDAP server, and restrict access at the firewall level.
303.5 LDAP server performance tuning	2	Measure the performance of your LDAP server, and tune for maximum performance.
303.6 OpenLDAP daemon configuration	2	Understand the basic slapd.conf configuration directives, and become familiar with the basic `slapd` command-line options.

Prerequisites

To get the most from this tutorial, you should have advanced knowledge of Linux and a working Linux system on which to practice the commands covered.

If your fundamental Linux skills are a bit rusty, you may want to first review the tutorials for the LPIC-1 and LPIC-2 exams.

Different versions of a program may format output differently, so your results may not look exactly like the listings and figures in this tutorial.

System requirements

To follow along with the examples in these tutorials, you need a Linux workstation with the OpenLDAP package and support for PAM. Most modern distributions meet these requirements.

Section 2. Access control lists in LDAP

This section covers material for topic 303.2 for the Senior Level Linux Professional (LPIC-3) exam 301. This topic has a weight of 2.

In this section, learn how to:

- Plan LDAP access control lists
- Understand access control syntax
- Grant and revoke LDAP access permissions

A variety of data can be stored in an LDAP tree, including phone numbers, birth dates, and payroll information. Some of these may be public, and some may be accessible by only certain people. The same information may have different restrictions based on the user. For example, perhaps only the owner of the record and administrators can change a phone number, but everyone can read the number. Access control lists (ACLs) handle the configuration of these restrictions.

Planning LDAP access control lists

Before you start writing your configuration, you should determine what you want to achieve. Which parts of the tree contain sensitive information? Which attributes need to be protected, and from whom? How will the tree be used?

The components of an ACL

An ACL entry supplies three pieces of information:

1. **What** entries and attributes the ACL specifies
2. **Who** the ACL applies to
3. The level of **access** that is granted

Regular expressions

Regular expressions are used for matching text. You may have a general idea of what the text looks like, or know certain patterns, and you can build a regular expression to find what you're looking for. A regular expression, or *regex* for short, consists of literal matches and meta characters that match a variety of patterns.

A simple regex is `hello`, which matches any string of characters containing the pattern `hello`.

You may not know if the string is capitalized, so the set meta characters, `[]`, match one occurrence of any of the characters inside the set. So, `[Hh]ello` matches both `hello` and `Hello`.

The period matches any single character. `.ello` matches `Hello` and `hello`, but also `fellow` and `cello`. It doesn't, however, match `ello`, because the period has to match something.

The characters `?`, `*`, and `+` match zero or one of the preceding character, zero or more of the preceding character, and one or more of the preceding character, respectively. Thus, `hello+` matches `hello` and `hellooooo`, but not `hell`.

Regular expressions have many different options and allow you to efficiently pull patterns from text files. In the context of OpenLDAP, regular expressions are used to match parts of a DN to avoid having to hand-code hundreds of different possibilities.

When specifying the "what" clause, you can choose to filter on the distinguished name (DN) of the object, an LDAP-style query filter, a list of attributes, or a combination of the three. The simplest clause allows everything, but you can get much more restrictive. Filtering on DN lets you specify an exact match, such as `ou=People,dc=ertw,dc=com`, or a regular expression (see "Regular expressions"). The query filter can match a certain `objectClass` or other attributes. The attribute list is a comma-separated list of attribute names. A more complex matching criteria might be "All password entries under

`ou=People,dc=ertw,dc=com` who are administrators."

You have a great deal of flexibility when determining who the ACL applies to. Users are generally identified by the DN with which they bind to the tree, which is called the *bindDN*. Each LDAP entry can have a `userPassword` attribute that is used to authenticate that particular user. In some contexts, you can refer to the currently logged-in user as *self*, which is useful for allowing a user to edit his or her own details.

If a user doesn't bind, they're considered *anonymous*. By default, anonymous users can read the tree, so you must decide if you need to change this behavior. You can further segment your anonymous users (or any user, for that matter) by IP address or the method used to connect (such as plaintext or encrypted).

Once you've determined the what and the who, you must determine the level of access. Access can range from *none* up to *write* access. You may also specify that the user can authenticate against the entry but can't read; or you can give the user read, search, and compare access.

Regardless of how you configure your ACLs, any configured `rootDN` users have full control over their database. You can't change this, except by removing the `rootDN` configuration from slapd.conf.

Understanding access control syntax

The basic form of an ACL, expressed in Backus-Naur Form, is:

```
access to <what> [ by <who> [ <access> ] [ <control> ] ]+
```

Backus-Naur Form

Backus-Naur Form (BNF) is a way to describe grammars such as ACL syntax. It's often used in developing Internet protocols because BNF is terse and very precise.

In BNF notation, you have a left-hand item and a right-hand item separated by a ::= symbol. This means the left-hand side can be substituted by items on the right-hand side. Items on the right-hand side that are enclosed in angle brackets (< and >) refer to another line of BNF, with the item in the angle brackets appearing on the left-hand side.

Items in square brackets ([and]) are optional. Vertical bars (|) indicate "one or the other"; and + and * mean "one or more of the preceding" and "zero or more of the preceding," respectively. Those familiar with regular expressions will recognize many of these items.

Looking at the BNF for an ACL, an ACL entry consists of the literal string "access to", followed by an item called "what" that is defined somewhere else. Following that are one or more lines of the form by <who> [<access>] [<control>], where who, access, and control are defined elsewhere, and both access and control are optional.

We explore the missing grammar in the rest of this tutorial.

Describe the what

The *what* describes the attributes and entries that are to be enforced by the ACL. The BNF notation to do so is shown in Listing 1.

Listing 1. BNF description of the what part of an ACL

```
<what>        ::= * |
    [dn[.<basic-style>]=<regex> | dn.<scope-style>=<DN>]
    [filter=<ldapfilter>] [attrs=<attrlist>]
<basic-style> ::= regex | exact
<scope-style> ::= base | one | subtree | children
<attrlist>    ::= <attr> [val[.<basic-style>]=<regex>]
    | <attr> , <attrlist>
<attr>        ::= <attrname> | entry | children
```

Some of the elements in Listing 1 aren't defined here, such as DN and regex. The form of the distinguished name is already known, and regular expressions are best studied outside of BNF.

Listing 1 shows that a description of the what portion of the ACL can be either the asterisk character (*), which matches everything, or a combination of a description of the DN, an LDAP filter, and a list of attributes. The latter possibility can use one or more of three components, because they're individually enclosed in square brackets.

Listing 2 shows three what clauses that match the DN.

Listing 2. Three sample what clauses

```
dn.exact="ou=people,dc=ertw,dc=com"
dn.regex="ou=people,dc=ertw,dc=com$"
dn.regex="^cn=Sean.*,dc=com$"
```

The first example matches only the entry for `ou=people,dc=ertw,dc=com`. If this ACL is used, it doesn't match any children such as `cn=Sean Walberg,ou=people,dc=ertw,dc=com`, nor does it match the parent entry.

The second example is similar to the first, but it uses a regular expression and *anchors* the search string with the dollar-sign ($) character. An anchor matches the position of the string rather than part of the string. The dollar sign matches the end of the string, so the second example matches anything ending in `ou=people,dc=ertw,dc=com`, which includes `cn=Sean Walberg,ou=people,dc=ertw,dc=com`. Note that without the anchor, the search string could be anywhere within the target, such as

`ou=people,dc=ertw,dc=com,o=MegaCorp`.

The third example from Listing 2 shows another anchor, `^`, which matches the beginning of the string. The third example also uses another regular expression, `.*`. The period matches any one character, and the asterisk matches zero or more of the preceding character. Thus, `.*` matches any string of zero or more characters. Put together, the third example matches any entry starting with `cn=Sean` and ending with `dc=com`.

You can also filter based on LDAP queries, the most helpful being a search on `objectClass`. For example, a what clause of `filter=(objectClass=posixAccount)` matches only entries with an `objectClass` of `posixAccount`. For a review of `objectClass`, see the first tutorial in this series, LPI exam 301 prep: Concepts, architecture, and design.

The final option for the what clause is to specify attributes. The most common usage is to restrict who can see private attributes, especially passwords. Use `attrs=userPassword` to match the password attribute.

Once you've determined what entries and attributes are to be matched, you must then describe who the rule will apply to.

Describe the who

Access is applied to a user, based on the DN that was provided at the time the client bound to the tree. The DN is usually found on the tree, but it could also be the `rootDN` provided in slapd.conf.

Listing 3 shows the BNF for notation for the who part of the ACL.

Listing 3. BNF notation for matching the who part of an ACL

```
<who> ::= * | [anonymous | users | self[.<selfstyle>]
                        | dn[.<basic-style>]=<regex> |
dn.<scope-style>=<DN>]
                  [dnattr=<attrname>]
[group[/<objectclass>[/<attrname>][.<basic-style>]]=<regex>]
                       [peername[.<peernamestyle>]=<peername>]
                  [sockname[.<style>]=<sockname>]
                  [domain[.<domainstyle>[,<modifier>]]=<domain>]
                  [ssf=<n>]
                  [transport_ssf=<n>]
                  [tls_ssf=<n>]
                  [sasl_ssf=<n>]

<style> ::= {exact|regex|expand}
<selfstyle> ::= {level{<n>}}
<dnstyle> ::= {{exact|base(object)}|regex
                  |one(level)|sub(tree)|children|level{<n>}}
<groupstyle> ::= {exact|expand}
<peernamestyle> ::= {<style>|ip|path}
<domainstyle> ::= {exact|regex|sub(tree)}
<modifier> ::= ={expand}
```

As in the what part of the ACL, an asterisk matches everything. To get more specific, you have many options. OpenLDAP defines three shortcuts named `anonymous`, `users`, and `self`. These shortcuts match unregistered users, authenticated users, and the currently logged-in user, respectively. The latter, `self`, is often used to allow the logged-in user to edit components of his or her own profile. This is based on an exact match of the DN; if you have a user's information split across different entries, the `self` keyword applies only to the entry the user bound with.

An interesting thing about the `self` keyword is that you can also make the ACL apply to parents or children of the user's entry with the `level` keyword. Using `self.level{1}` matches the user's entry and the parent entry, whereas `self.level{-1}` matches the user's entry and any directly attached children.

Still looking at the DN, you can perform regular-expression or exact matches with `dn.exact="`*`DN`*`"` and `dn.regex="`*`regex`*`"`, respectively. A later example will show how to use regular expressions to dynamically tie the what and the who together.

Arbitrary entries can be protected using the `dnattr` keyword, which also requires the name of an attribute. If the DN of the requester appears in the specified attribute of the target, the ACL is matched. For example, if you add `dnattr=manager` to your ACL and then add `manager: cn=Joe Blow,ou=people,dc=ertw,dc=com` to Fred Smith's entry, the ACL will match when Joe Blow accesses Fred Smith's entry.

The `group` keyword is similar to `dnattr`, except that the parameters refer to a group defined elsewhere in the tree rather than an attribute in the entry. By default, the group has an `objectClass` of `groupOfNames`, and the members are referenced in the `member` attribute.

Use the `peername`, `sockname`, and `domain` keywords to match attributes of the client connection. `peername` refers to the IP address of the client, such as `peernameip=127.0.0.1`. `sockname` is for connections over named pipes, which aren't commonly used. `domain` matches the hostname associated with the IP address, which can be easily spoofed.

The final set of options refers to the Security Strength Factor (SSF) of the connection, which is an OpenLDAP term for the connection's level of security. These options will become clearer when you're introduced to the security mechanisms used to connect to OpenLDAP, such as Transport Layer Security (TLS) and Simple Authentication and Security Layer (SASL).

All of the preceding items can be used together. For example, you could allow write access to the password field only to certain administrators coming from a certain IP address range with a certain level of encryption. You could also do far less, such as

requiring only a valid login, or even accepting everyone regardless of authentication.

Describe the access

Once you've determined who is accessing your tree and what they're trying to access, you must specify what level of access they have. Listing 4 shows the BNF notation for the access part of the ACL.

Listing 4. BNF notation describing the format of the access clause

```
<access>  ::=  [[real]self]{<level>|<priv>}
<level> ::= none|disclose|auth|compare|search|read|write
<priv> ::= {=|+|-}{w|r|s|c|x|d|0}+
```

When specifying access using the `level` format, each successive level includes the ones before it. That is, `read` access gives `search`, `compare`, `auth`, and `disclose` access. `none` and `disclose` both deny any access, except that some error messages that might disclose information about the contents of the tree are removed under `none` and allowed under `disclose`.

Alternatively, you can specify the level of access in terms of the LDAP operations permitted using the `priv` format. The options run opposite to the `level` format, such that `w` is for write and `0` is for none. When specifying access using the `priv` format, there is no implied progression as is the case with `level`. If you want to offer full access, you must do so with `wrscx`.

The `=/+/-` symbol before the letters denotes how the specified access is merged with the current access level if multiple rules apply. With `=`, all previously defined access is ignored, and the value to be used follows the equal sign. With `+` and `-`, access is added to or subtracted from the current level, respectively.

Understand control

By default, OpenLDAP takes a first-match approach to applying access lists. OpenLDAP finds the first ACL entry that matches the what clause and, within that entry, finds the first entry matching the who part. This is the same as putting the keyword `stop` after the access level is described. The other two options are `continue` and `break`. If you use `continue`, the current ACL entry is searched for the next line matching the who part. If you use `break`, processing of the current ACL entry stops, but OpenLDAP looks for the next ACL entry matching the who clause.

Pulling together the what, who, and access

Now that you've seen the three (four, if you count control) parts of the ACL, you can bring them together into a policy. Listing 5 shows a typical list of ACLs that allows

registered users to read the tree and lets users update their own passwords (but not read them).

Listing 5. A simple ACL setup

```
access to attrs=userPassword
    by self =xw
    by anonymous auth

access to *
    by self write
    by users read
```

The first clause matches anyone trying to access the `userPassword` field. The user is given write and authentication permission to their own entry, which is enforced through the equals sign. Anonymous users are given authentication permission. A user is anonymous as they're binding to the tree; therefore, anonymous users require the `auth` permission so they may log in to be a regular, privileged user.

If the information being requested isn't the password, the second ACL entry is consulted. Again, the user has full control over his or her own entry (except the `userPassword` field, by virtue of the first ACL entry), whereas all authenticated users have read access to the rest of the tree.

Listing 6 shows an ACL entry that uses regular expressions to tie the what and who clauses together.

Listing 6. Getting fancy with regular expressions

```
access to dn.regex="cn=([^,]+),ou=addressbook,dc=ertw,dc=com"
        by dn.regex="cn=$1,ou=People,dc=ertw,dc=com" write
        by users read
```

Listing 6 allows users to edit their corresponding records under the `ou=addressbook,dc=ertw,dc=com` part of the tree. `[^,]+` matches everything up to, but not including, a comma, and the parentheses save the matched text into `$1` for the first set of parentheses, `$2` for the next, and so forth up to and including `$9`. The who clause reuses the name of the user to determine who can access the entry. If the name of the user is the same as that of the entry being accessed, then the user is given write access. Failing that, authenticated users are given read access.

Practical considerations

It's wise to keep the more specific ACL entries at the top of the list because of the first-match behavior; otherwise, it's more likely that a previous ACL will cause the

one lower down the list to be ignored. This technique can also be used to grant and revoke access to a particular user. Simply put your specific ACL clause at the top of the list.

Keep your ACLs as simple as possible. Doing so reduces the possibility of error and also improves performance. ACLs must be parsed each time an operation is carried out against the tree.

Section 3. LDAP replication

This section covers material for topic 303.3 for the Senior Level Linux Professional (LPIC-3) exam 301. This topic has a weight of 5.

In this section, you learn how to do the following:

- Understand replication concepts
- Configure OpenLDAP replication
- Execute and manage slurpd
- Analyze replication log files
- Understand replica hubs
- Configure LDAP referrals
- Configure LDAP sync replication

At some point, your needs may extend beyond one server. Your organization may rely on LDAP to the extent that the loss of your LDAP server is unacceptable, or your query volume may be high enough that you have to split your queries across multiple servers. It could even be a combination of both; but in any case, you need to use more than one server.

With multiple servers, you can partition your tree across different servers, but this leads to a decrease in reliability -- not to mention that it may be difficult to properly balance your queries. Ideally, each server has an identical copy of the tree. Any writes are propagated to the other servers in a timely fashion so that all the other servers are up to date. This is called *replication*.

This scenario, called *multi-master*, is complex because the data has no clear single owner. Most often, a master-slave relationship is formed, where one server takes on all the writes and sends them to the slaves. LDAP queries can be made against any

server. This can be extended to the *replica hub* scenario, where a single slave server replicates data from the master and, in turn, replicates data to several other slaves.

OpenLDAP provides two methods to achieve replication. The first is through *slurpd*, a separate daemon that watches for changes on the master and pushes the changes to the slaves. The second is using slapd's LDAP sync replication engine, otherwise known as *syncrepl*. The slurpd method is now considered obsolete; people are urged to use syncrepl instead. Both these methods are investigated in this section.

slurpd-based replication

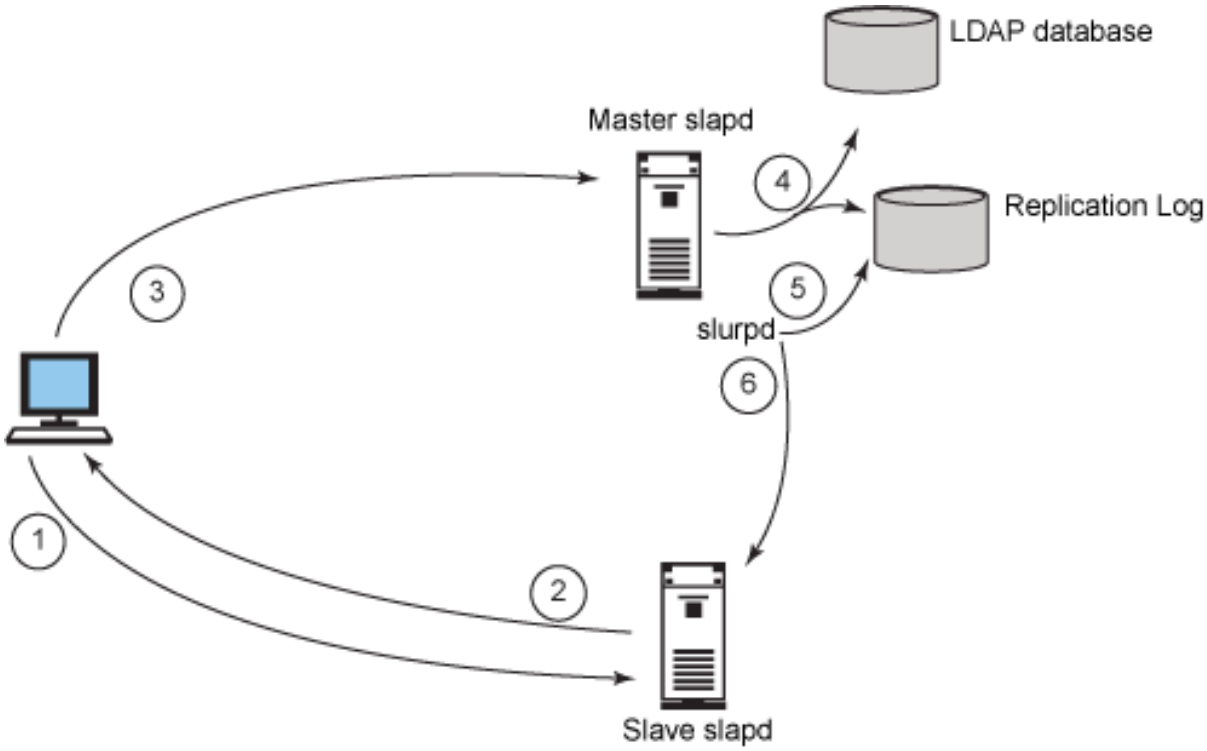
Slurpd-based replication is a push replication in that the master pushes any changes to the slaves. If a client attempts to update a slave, the slave is configured to send a *referral* back to the client, pointing the client to the master. The client is responsible for reissuing the request to the master. Slurpd is a standalone daemon that is configured from slapd.conf.

Data flow within the slurpd replication model

The master server is the server that handles all the writes from the clients and holds the authoritative source of data. Any changes to the master's tree are written to a *replication log*, which is monitored by slurpd. slurpd pushes the changes to all the slaves upon noticing a change in the replication log.

Figure 1 shows the typical behavior of slurpd.

Figure 1. Data flow in the slurpd replication model



The process is as follows:

1. The client sends an update request, which happens to be received by a slave.
2. The slave knows that writes can only come from its replication partner, and therefore it sends a referral back the client, pointing it to the master server.
3. The client reissues the update request to the master.
4. The master performs the update and writes the change to the replication log
5. slurpd, also running on the master, notices the change in the replication log.
6. slurpd sends the change to the slave.

In this way, slaves can be kept up to date with the master with little lag. If any interruptions happen, or an error occurs on a slave, slurpd always knows which slaves need which updates.

Configure slurpd

Configuring slurpd-based replication require the following steps:

1. Create a replica account that slurpd will use to authenticate against the slave replica.
2. Configure the master server with the name of the slave.
3. Configure the slave to be a replica, including any needed ACLs.
4. Copy the database from the mater to the slave.

Creating the replica is straightforward; the only requirement is that the account must have a password in the `userPassword` attribute. You may use the `inetOrgPerson` `objectClass` like most accounts belonging to people, or you can use a more generic `objectClass` such as `account` with the `simpleSecurityObject` auxiliary `objectClass` added. Recall from the first tutorial that structural `objectClasses` define the entry (and therefore you may use only one per entry), whereas auxiliary `objectClasses` add attributes to any entry regardless of the structural `objectClass`. Listing 7 shows the LDIF code to add a replica account.

Listing 7. LDIF code for a replica account

```
dn: uid=replica1,dc=ertw,dc=com
uid: replica1
userPassword: replica1
description: Account for replication to slave1
objectClass: simpleSecurityObject
objectClass: account
```

Listing 7 shows a sparse entry -- just a username, password, and description, which is adequate for replication. The description is optional, but it's recommended for documentation. Remember the password; you'll need it in the next step!

The master must now be configured to store all changes in a replication log, and a replica must be configured for slurpd to work. It's important to remember that slurpd reads its configuration from slapd.conf, and that slurpd pushes the updates to the slaves. This will help you remember where to configure replication and that the authentication credentials belong on the master. The slave can validate the credentials because the account is part of the tree. Listing 8 shows the configuration on the master to enable replication.

Listing 8. Configuration of master for slurpd replication

```
replica  uri=ldap://slaveserver.ertw.com
    suffix="dc=ertw,dc=com"
    binddn="uid=replica1,dc=ertw,dc=com"
```

```
    credentials="replica1"
    bindmethod=simple

replogfile /var/tmp/replicationlog
```

Configuration of replication happens in database mode, so be sure to have the `replica` command somewhere after your first `database` configuration. The `replica` takes several parameters of the form `key=value`. The `uri` key specifies the name or IP address of the slave in uniform resource identifier (URI) format, effectively the name of the slave prepended with `ldap://`.

After specifying the name of the slave, you can optionally configure the name of the database to replicate through the `suffix` option. By default, all databases are replicated. The final requirement is to provide authentication information so that slurpd can connect to the specified `uri`. For simple authentication, the `binddn`, `bindmethod`, and `credentials` (the `userPassword` you assigned earlier) are all you need.

The final piece of the puzzle is to tell slapd where to store its replication log. You need to provide a full path because relative filenames don't work. Don't worry about creating the file, because slapd will create it for you; but the path you specify must be writable by the user slapd and slurpd are running as.

On the slave server, you must configure the replication account and also tell the slave that it should return a referral back to the master for any writes. Listing 9 shows this configuration.

Listing 9. Configuration for the slave

```
updatedn uid=replica1,dc=ertw,dc=com
updateref ldap://masterserver.ertw.com
```

The `updatedn` refers back to the account you created earlier on the master that slurpd will use to push the updates to the slaves. The `updateref` is another LDAP URI, this one pointing to the master. Listing 10 shows a client trying to update the slave after the previous configuration has been loaded, and receiving a referral.

Listing 10. A client receiving a referral after trying to update a slave

```
[root@slave openldap]# ldapadd -x -D cn=root,dc=ertw,dc=com -w mypass -f
newaccount.ldif
adding new entry "cn=David Walberg,ou=people,dc=ertw,dc=com"
ldap_add: Referral (10)
        referrals:
ldap://masterserver.ertw.com/cn=David%20Walberg,ou=People,dc=ertw,dc=com
```

The OpenLDAP command-line clients don't follow referrals, but other LDAP libraries

do. If you're using LDAP in a replicated environment, you should verify that your applications follow referrals correctly.

The final piece of the puzzle is to make sure the master and slave start with identical databases. To do so, follow these steps:

1. Shut down the master server.
2. Shut down the slave server.
3. Copy all the database files from the master to the slave.
4. Start both the master and the slave servers.
5. Start up slurpd.

Both servers must be shut down to copy the database, to ensure that no changes are made and that all data has been written to disk. It's vital that both servers start with the same data set; otherwise they may get out of sync later. slurpd-based replication essentially plays back all the transactions on the slave that happened on the master, so any differences can cause problems.

slurpd may automatically start up with slapd, depending on your distribution and start-up scripts. If it hasn't started automatically, start it by running `slurpd` at the command line.

At this point, replication should be running. Create an account on your master, and test. Also test that your slave sends back referrals if you try to update it.

Monitor replication

Understanding how to monitor replication is important because errors can happen that may cause data to get out of synchronization. The same skills in monitoring also help in debugging.

slurpd stores its own files in /var/lib/ldap/replica (this is separate from the replication log that is produced by slapd). In this directory are slurpd's own replication logs and any *reject files*. If slurpd tries to send an update to a slave that fails, the data is stored in a file named after the slave, with an extension of .rej. Inside the file is the LDIF code making up the entry along with the error returned by the slave, such as `ERROR: Already exists`. Listing 11 shows the contents of a .rej file with a different error.

Listing 11. A replication rejection file

```
ERROR: Invalid DN syntax: invalid DN
```

```
replica: slaveserver.ertw.com:389
time: 1203798375.0
dn: sendmailMTAKey=testing,ou=aliases,dc=ertw,dc=com
changetype: add
objectClass: sendmailMTAAliasObject
sendmailMTAAliasGrouping: aliases
sendmailMTACluster: external
sendmailMTAKey: testing
sendmailMTAAliasValue: testinglist@ertw.com
structuralObjectClass: sendmailMTAAliasObject
entryUUID: 5375b66c-7699-102c-822b-fbf5b7bc4860
creatorsName: cn=root,dc=ertw,dc=com
createTimestamp: 20080223202615Z
entryCSN: 20080223202615Z#000000#00#000000
modifiersName: cn=root,dc=ertw,dc=com
modifyTimestamp: 20080223202615Z
```

The rejection file in Listing 11 starts with a text error ("ERROR: Invalid DN syntax: invalid DN"), and the rest looks like LDIF. Note the first attribute is **replica**, which is the name of the replica that couldn't process the update, and the second attribute, **time**, is the time the error occurred (in UNIX timestamp format). The next few attributes come from the entry that was rejected.

The last 7 attributes are called **operational attributes**. They weren't part of the original change, but were added by the LDAP server for internal tracking. A universally unique identifier (UUID) was given to the entry, along with some information on when and who changed the record.

In Listing 11, the error most likely comes from a missing schema on the slave. The slave doesn't understand what the **sendmailMTAKey** attribute is, therefore the DN of the entry is invalid. The slave must have its schema updated before replication can continue.

To fix a rejected entry, you must evaluate the error and fix the underlying problem with the tree. Once you know the rejected entry will apply cleanly, use slurpd's *one-shot mode* to apply the update with `slurpd -r /path/to/rejection.rej -o`. The `-r` parameter tells slurpd to read from the given replication log, and `-o` causes slurpd to use one-shot mode, meaning it exits after processing the log instead of the default behavior of waiting for more entries to be added to the log.

If replication isn't working at all, the best approach is to start with the master and work your way to the slave. First, kill slurpd and make a change to the tree. Then, check to see if the replication log is growing; if it isn't, the master is set up incorrectly. Next, start up slurpd, and pass `-d 255` on the command line. This traces slurpd's actions as it processes the logs. Look for errors, especially related to opening files and access controls.

Finally, on the slave, use `loglevel auth sync` to check for any errors when replication is happening (slapd logs to syslog with the `local4` facility, so you may need to add `local4.* /var/log/slapd.log` to /etc/syslog.conf).

LDAP Sync replication

slurpd is a straightforward solution to the replication problem, but it has several shortcomings. Shutting down your master server so that you can synchronize a slave is inconvenient at best, and at worst it can affect service. The push architecture of slurpd can also be limiting. slurpd worked well for its time, but something better had to be created. RFC 4533 outlines the LDAP Content Synchronization Operation, which is implemented in OpenLDAP by the LDAP Sync replication engine, otherwise known as *syncrepl*.

syncrepl is built as an overlay that is inserted between the core of slapd and the backend database. All writes to the tree are tracked by the syncrepl engine rather than requiring a separate server instance. Other than the mechanics of the replication and the roles (described next), the concepts are similar to slurpd. Writes to a replica are refused, with a referral passed back to the master server.

syncrepl is initiated from the slave, which is now given the name *consumer*. The master role is called *provider*. In syncrepl, the consumer connects to the provider to get updates to the tree. In the most basic mode, called *refreshOnly*, the consumer receives all the changed entries since its last refresh, requests a cookie that keeps track of the last synchronized change, and then disconnects. On the next connection, the cookie is presented to the provider, which sends only the entries that changed since the last synchronization.

Another syncrepl mode, called *refreshAndPersist*, starts off like the refreshOnly operation; but instead of disconnecting, the consumer stays connected to receive any updates Any changes that happen after the initial refresh are immediately sent over the connection to the consumer by the provider.

Configure syncrepl

Listing 12 shows the provider's configuration for both syncrepl modes (refreshOnly and refreshAndPersist).

Listing 12. Provider configuration for syncrepl

```
overlay syncprov
syncprov-checkpoint 100 10
syncprov-sessionlog 100
```

The first line of Listing 12 enables the syncprov overlay. Overlays must be configured against a database; therefore, this configuration must go after your `database` configuration line. The next two lines are optional, but they improve reliability. `syncprov-checkpoint 100 10` tells the server to store the value of `contextCSN` to disk every 100 write operations or every 10 minutes. `contextCSN`

is part of the cookie mentioned earlier that helps consumers pick up where they left off after the last replication cycle. `syncprov-sessionlog 100` logs write operations to disk, which again helps in the refresh cycle.

More details about configuring the provider are found in the `slapo-syncprov(5)` manpage.

Listing 13 shows the consumer side of the replication pair.

Listing 13. Consumer configuration for syncrepl in refreshOnly mode

```
updateref ldap://masterserver.ertw.com
syncrepl rid=1
 provider=ldap://masterserver.ertw.com
 type=refreshOnly
 interval=00:01:00:00
 searchbase="dc=ertw,dc=com"
 bindmethod=simple
 binddn="uid=replica1,dc=ertw,dc=com"
 credentials=replica1
```

Like the `replica` command from the slurpd configuration, the `syncrepl` command requires an `updateref`, information about the tree you're trying to replicate, and the authentication credentials you're going to use. This time, the credentials go on the consumer side and require enough access on the provider to read the part of the tree being replicated. The updates to the database on the consumer are run as the `rootdn`.

Items in Listing 13 specific to syncrepl are the `rid`, `provider`, `type`, and `interval`. The `rid` identifies this consumer to the master. The consumer must have a unique ID between 1 and 999. The `provider` is an LDAP URI pointing back to the provider. `type` specifies that you only want periodic synchronization through `refreshOnly`, and the `interval` is every hour. The `interval` is specified in `DD:hh:mm:ss` format.

Start the consumer with an empty database, and it will replicate its data from the provider and update every hour.

Making the transition to refreshAndPersist mode is simple. In Listing 13, remove the `interval`, and change the `type` to `refreshAndPersist`

syncrepl Filtering

It's worth noting that you don't have to replicate the whole LDAP tree. You can use the following commands to filter the data that is replicated.

Table 3. Commands for filtering replication traffic

Command	Description

searchbase	A DN pointing to the node in the tree where replication will start. OpenLDAP will fill in any necessary parent nodes to make the tree complete.
`scope`	One of `sub`, `one`, or `base`. This determines how far down the tree, starting at the `searchbase`, that data is replicated. The default is `sub`, which covers the `searchbase` and all children
`filter`	A LDAP search filter, such as `(objectClass=inetOrgPerson)` that controls which records are replicated.
`attrs`	A list of attributes that will be copied over from the selected entries.

Like the other options for `syncrepl`, these options are entered in the form of `key=value`

Section 4.

This section covers material for topic 303.4 for the Senior Level Linux Professional (LPIC-3) exam 301. This topic has a weight of 4.

In this section, you learn how to do the following:

- Secure the directory with SSL and TLS
- Configure and generate client / server certificates
- Understand firewall considerations
- Configure unauthenticated access methods
- Configure User / password authentication methods

Up to this point, all access to slapd has been over unencrypted channels using cleartext passwords. This is called *simple* authentication. This section looks at adding encryption to the client-server connection.

Use SSL and TLS to secure communications

You may be familiar with Secure Sockets Layer (SSL) and Transport Layer Security (TLS) as the protocols that secure Web transactions. Whenever you browse an https type URI, you're using SSL or TLS. TLS is an improvement on SSLv3 and in some cases is backward compatible to SSL. Because of their shared heritage and compatibility, they're often referred to collectively as SSL.

SSL makes use of X.509 certificates, which are a piece of data in a standard format that has been digitally signed by a trusted third party known as a Certificate Authority (CA). A valid digital signature means the data that was signed hasn't been tampered with. If the data being signed changes, even by one bit, then the signature won't validate. Independent parties, such as the client and the server, can validate signatures because they both start off by trusting the CA.

Within the server's certificate is information about the ownership of the server, including its name on the Internet. Thus you can be sure you're connecting to the right server because the name of the server you connected to exactly matches the name in the certificate, and you've trusted the CA to validate this before signing. The certificate also includes the server's public key, which can be used to encrypt data such that only the holder of the secret key can decrypt it.

Public and secret keys form the basis of *public key* or *asymmetric* cryptography. It's asymmetric because data encrypted by the public key can only be decrypted by the secret key, and data encrypted with the secret key can only be decrypted by the public key. For what you normally think of as encryption, such as keeping a message secret, the first case is used. The public key is made public, and the secret key is made secret. Because of the asymmetric behavior of the keys, the secret key can encrypt a message, and anyone with the public key can decrypt it. This is how digital signatures work.

After the client connects to the server and receives the server's certificate, the client can validate that the server name is correct, which prevents a *man in the middle attack*. The public key can be used to run through a protocol that ends up with the client and server agreeing on a shared secret that no one observing the conversation can determine. This secret is then used to encode the rest of the conversation between the client and the server, called *symmetric* cryptography because the same key both encrypts and decrypts the data. The divide between asymmetric and symmetric cryptography exists because the latter is orders of magnitude faster. Public key cryptography is used to authenticate and come up with a shared secret, and then symmetric key cryptography takes over.

To apply this all to OpenLDAP, you must create a certificate for the server and then configure the server to use it. This example uses a self-signed certificate rather than creating a Certificate Authority, which means the final certificate has been signed by itself. It doesn't provide the level of trust that a CA does, but it's adequate for testing. Listing 14 shows the generation of the key.

Listing 14. Generating the TLS key pair

```
[root@bob ssl]# openssl genrsa -out ldap.key 1024
Generating RSA private key, 1024 bit long modulus
.............................++++++
.....................++++++
e is 65537 (0x10001)
```

Listing 14 shows the key being generated with the `openssl genrsa` command. The key is 1024 bits long, which is currently considered adequate for public keys (note that using much larger values makes cryptographic operations much slower and may confuse some clients). Next, `openssl req` takes the public part of the freshly generated key pair, adds some location information, and packages the result -- a Certificate Signing Request (CSR) -- to be signed by a CA (see Listing 15).

Listing 15. Generating the Certificate Signing Request

```
[root@bob ssl]# openssl req -new -key ldap.key -out ldap.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a
DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:CA
State or Province Name (full name) [Berkshire]:Manitoba
Locality Name (eg, city) [Newbury]:Winnipeg
Organization Name (eg, company) [My Company Ltd]:ERTW
Organizational Unit Name (eg, section) []:Directory Services
Common Name (eg, your name or your server's hostname)
[]:masterserver.ertw.com
Email Address []:sean@ertw.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

The generated file, ldap.csr, can be sent off to a CA (along with a hefty payment) to be signed. This is also the procedure you follow to generate a certificate for a Web server. If you do send this off for signing, make sure all the information you've provided is spelled correctly, that abbreviations are used only for the Country Name, and that the Common Name *exactly* matches the DNS name people will use to connect to your server.

Instead of getting a CA to sign the CSR, in this example you'll sign it yourself, as shown in Listing 16.

Listing 16. Signing the CSR

```
[root@bob ssl]# openssl x509 -req -days 1095 -in ldap.csr -signkey
```

```
ldap.key -out ldap.cert
Signature ok
subject=/C=CA/ST=Manitoba/L=Winnipeg/O=ERTW/OU=Directory Services/
  CN=masterserver.ertw.com/emailAddress=sean@ertw.com
Getting Private key
```

Listing 16 signs the key with the `openssl x509` command. Using `-req` tells openssl that the input is a CSR. The validity of this certificate is 1095 days, or 3 years. Now you have ldap.key (the private key) and ldap.cert (the certificate and public key).

Before continuing, add a line to /etc/openldap/ldap.conf containing `TLS_REQCERT allow`. This tells the OpenLDAP client utilities to ignore the fact that they're seeing a self-signed certificate. Otherwise, the default settings deny the certificate as invalid.

Getting OpenLDAP to use the new key and certificate is easy. Assuming you stored the generated keys in /etc/openldap/ssl/, the lines in Listing 17 set up your server for TLS connections after you restart slapd.

Listing 17. Configuring slapd for SSL

```
TLSCertificateFile /etc/openldap/ssl/ldap.cert
TLSCertificateKeyFile /etc/openldap/ssl/ldap.key
```

The commands in Listing 17 point slapd to your certificate and private key. To test your new server, issue the `ldapwhoami -v -x -Z` command, which binds anonymously to your secure port. If you receive a "success" message, then everything is working correctly. Otherwise, the debugging information generated by `-v` will point you to the cause or any errors.

You can generate a client certificate, which is optional, the same way as the server certificate. Instead of Listing 17, use the `TLS_KEY` and `TLS_CERT` commands in ldap.conf, which set your client key and certificate, respectively. Client certificates are required only if you need to have the certificate itself identify the client.

Firewall considerations

LDAP uses TCP port 389, and LDAPS (LDAP over SSL) uses TCP port 636. If you have a firewall between your servers and your clients, these ports must be allowed through the firewall for connections to succeed. Clients always connect to servers, and, depending on your replication strategy, servers connect to other servers.

Linux iptables

If you have an iptables-based firewall on your LDAP server, you need to modify your rule set to allow the incoming connections. Generally, the commands in Listing 18

suffice.

Listing 18. Adding iptables rules to allow LDAP connections

```
iptables -A INPUT -p tcp --dport 389 -j ACCEPT
iptables -A INPUT -p tcp --dport 636 -j ACCEPT
```

Listing 18 works if your policy is simple. `-A INPUT` appends the rule to the INPUT table, where all incoming packets are checked. You may have to insert these rules at the top (use `-I INPUT` instead) or use your distribution's firewall tools to allow TCP ports 389 and optionally 636 if you need LDAPS connectivity.

If you're using your Linux firewall as a router, such that the clients are attached to another interface and the LDAP server is attached to another interface, you must use the `FORWARD` chain instead of `INPUT`. You may also want to specify the incoming interface with `-i`, such as `-i eth0` to indicate that only packets coming in eth0 will be accepted. Once a packet has been accepted, the return packets are also accepted.

Protection through TCP Wrappers

One of the configuration options available when compiling OpenLDAP is `--enable-wrappers`, which links the resulting binaries against the TCP Wrappers libraries. The wrappers use two files, /etc/hosts.allow and /etc/hosts.deny, to permit or deny access to incoming clients.

First check to see if slapd uses TCP Wrappers with `ldd /usr/sbin/slapd | grep libwrap`. If anything is returned, then your binary is using TCP Wrappers. If not, you need to recompile with `--enable-wrappers` or use the iptables method shown earlier.

With wrappers support, you can deny everybody access by adding `slapd: ALL` in /etc/hosts.deny. You can then allow people in with `slapd: 192.168.1.,127.0.0.1`, which lets anyone from the 192.168.1.0/24 network or the localhost connect. Note that connections denied through TCP Wrappers connect at first but are then disconnected automatically. Contrast this to a firewall, where the packet is dropped before it reaches slapd.

The format of hosts.allow and hosts.deny allows for many different ways to allow and deny connections; consult the hosts_access(5) manpage for all the details.

More on authentication

So far, the discussion of authentication has been limited to cleartext passwords defined in slapd.conf and simple authentication used between the client and the

server. Cleartext passwords can be solved with `slappasswd`. Enter `slappasswd` at the shell, and you're prompted for a password and then to verify that password. The output is a secure hash of the password, such as `{SSHA}oxmMsm9Ff/xVQ6zv+bgmMQjCUFL5x22+`. This hash is mathematically guaranteed not to be reversible, although given the hash, someone could guess repeatedly by trying various passwords and seeing if the hash is the same.

You've already experienced the anonymous bind, where no username or password has been provided, and the authenticated bind, where both the username and password are provided and valid. OpenLDAP also supports an unauthenticated bind, where the username is provided with no password. An unauthenticated bind is usually disabled unless you have `allow bind_anon_cred` in your configuration. If allowed, an unauthenticated bind is considered anonymous.

The alternative to simple authentication is Simple Authentication and Security Layer (SASL), a framework for providing a plug-in architecture for authentication and encryption methods. A detailed look at SASL will appear in a forthcoming tutorial; in the meantime, SASL allows for different authentication methods from plaintext to Kerberos.

Earlier, when investigating ACLs, this tutorial mentioned that access can be influenced by the connection method. This is called the *Security Strength Factor* (SSF). An unencrypted session has an SSF of 0, and an encrypted session generally has an SSF corresponding to the key length. Thus, you can require an encrypted session for a particular ACL by adding `ssf=1` to the who clause of your ACL.

Section 5. LDAP server performance tuning

This section covers material for topic 303.5 for the Senior Level Linux Professional (LPIC-3) exam 301. This topic has a weight of 2.

In this section, you learn how to do the following:

- Measure LDAP performance
- Tune software configuration to increase performance
- Understand indexes

OpenLDAP is a database. You provide it with a query or a task to run, and it finds the data and returns it to you. In order to do so as fast as possible, you must assign

resources to the places they will be best used, such as caching of frequently accessed data, and indexing databases.

Measure performance

Before you can try to make slapd run faster, you must be able to measure the current state. This may mean timing a certain operation in your application and looking for an improvement. It may also mean performing several queries yourself and calculating the average. The metric may not even be time based; it may be to reduce disk load on the LDAP server because the current configuration is causing too many reads and writes.

Either way, it's helpful to take several measurements of different metrics before and after any changes. Helpful commands are as follows:

- `vmstat` to show input/output (IO) statistics and CPU usage, notably the user time and wait time
- `iostat` to show more details about reads and writes to disk, along with the disk controller usage
- `ps` to show memory usage of the slapd process (not that using more memory is a bad thing, but it's important to make sure you don't run out of RAM)
- `time` for timing various command-line operations

Tune the daemon

Tuning always involves tradeoffs. Often, you increase the amount of resources (usually memory or disk) to a process to get the process to respond faster. Doing so decreases the resources that other processes can use. Similarly, if you make your process run faster, it often consumes more resources, such as CPU cycles or disk IO, that are unavailable to other processes.

Tradeoffs can also occur at the application level. By sacrificing some write performance, you're often able to drastically improve read performance You can also make your application run faster by turning off various safety features such as transaction logging. Should you have a crash, you may end up restoring your database from a backup, but only you know if this is an acceptable tradeoff.

Most people use the Berkeley Database (BDB) backend. This backend is based on the Sleepycat Berkeley Database, now owned by Oracle, which is a fast embedded database. It doesn't support a query language; instead, it's based on hash-table lookups. Tuning this backend occurs in two places: one in slapd.conf and another in

a special file used by the BDB runtime.

slapd.conf configuration directives

The BDB database is linked against the binary that uses it, rather than being a standalone server like most SQL servers. As such, the application using the BDB database is responsible for some of the behavior of the database. The slapd-bdb(5) manpage describes all the directives that are controllable by slapd through slapd.conf; this tutorial covers only the most important.

Like many SQL backends, BDB databases write their changes to a transaction log to ensure reliability in the case of a failure; they also keep data in memory to save on disk writes. The operation that flushes all the in-memory data to disk and writes the transaction log out is called a *checkpoint*. The `checkpoint` command tells slapd how often to write out the data, in terms of kbytes of stored changes and minutes since the last checkpoint. Adding `checkpoint 128 15` to slapd.conf means that data will be flushed every 128KB of changes or at least every 15 minutes. By default, no checkpoint operations are performed which is the same as `checkpoint 0 0`.

Recently accessed entries can be cached in RAM for faster access. By default, 1000 entries are cached. To change this number, use `cachesize` along with the number of entries. The higher the `cachesize`, the more likely an entry is to be cached in RAM, but the more RAM is consumed by the slapd process. Your choice of value here depends on how many different entries are in your tree and the pattern of access. Make sure you have enough space in the cache to hold your commonly accessed items, such as the user list.

Similar to `cachesize` is `idlcachesize`, which has to do with how big the memory cache for indexes is. Your setting will depend on how many indexes you have configured (discussed later), but it's wise to start by making `idlcachesize` the same as `cachesize`.

Tune the BDB databases

As mentioned earlier, some of the tuning parameters for the BDB databases are handled in a separate file that is read by the BDB runtime and ignored by slapd. This file is called DB_CONFIG, and it lives in the same directory as your database. The most important parameter in that file is `set_cachesize`, which sets the internal BDB cache size, not the slapd entry cache. The format is `set_cachesize <GigaBytes> <Bytes> <Segments;>`, where GigaBytes and Bytes refer to the size of the cache (the two are added together), and Segments allows you to split the cache across separate memory blocks to get around 32-bit address limitations (both 0 and 1 have the same effect, allowing only a single memory segment). For a 1GB cache, use `set_cachesize 1 0 0`.

To determine the best BDB cache size, it's often easiest to look at the cache statistics of a working system and increase as needed. The command to look at the memory usage statistics of a BDB database is `db_stat -h /path/to/database -m`. The first 20 lines show the relevant details. If you have a high number of pages forced from the cache, or the number of pages found in the cache is low (< 95%), consider increasing the BDB cache size. On some distributions, `db_stat` may be called `slapd_db_stat` to separate it from the system BDB libraries and tools.

In addition to cache, you need to make sure the transaction logs are monitored. Set the path to the transaction logs with `set_lg_dir`. If you can put the transaction log on a different set of disk spindles than the database, you'll have much better performance.

Even though BDB is a simple database, it still needs to be able to lock files for writing. The default settings for the number of locks is usually large enough, but you should monitor the output of `db_stat -h /path/to/database -c` for any mention of hitting the maximum number of locks. In BDB, locks are split into three types (and reported separately): lockers, locks, and lock objects The difference between them isn't important, but the maximum number of each is controlled through `set_lk_max_lockers`, `set_lk_max_locks`, and `set_lk_max_objects`, respectively.

Whenever you make changes to DB_CONFIG, you must restart slapd. Listing 19 shows a sample DB_CONFIG file based on the directives mentioned previously.

Listing 19. Sample DB_CONFIG file

```
# 256K cache
set_cachesize 0  268435456 0
set_lg_dir /var/log/openldap
set_lk_max_lockers 1000
set_lk_max_locks 1000
set_lk_max_objects 1000
```

Index your database

Most LDAP operations involve some sort of search on an attribute, such as a username, a phone number, or an email address. Without anything to help it, slapd must search each entry when performing a query. Adding an index to an attribute creates a file that lets slapd search much more quickly because the data in an index is stored in a way that allows for fast lookups. The tradeoff for an index is slower write speed and increased disk and memory usage. Therefore, it's best to index attributes that are searched on frequently.

OpenLDAP supports several index types, depending on the type of search being performed. The index types are listed in Table 4.

Table 4. OpenLDAP index types

Type	keyword	Description	Search example
Presence	pres	Used for lookups that want to know whether an attribute exists.	`uid=*`
Equality	eq	Used for lookups that are looking for a specific value.	`uid=42`
Substring	sub	Used for lookups that are looking for a string somewhere within a value. Within this type, you can specify three other optimized types or use the generic `sub` type.	`cn=Sean*`
	subinitial	A substring index looking for a string at the beginning of a value.	`cn=Sean*`
	subany	A substring index looking for a string in the middle of a value.	`cn=*jone*`
	subfinal	A substring index looking for a string at the end of a value.	`cn=*Smith`
Approximate	approx	Used for sound-alike searches, where you want to find a value that sounds like your search string.	`cn~=Jason`

To apply an index to an attribute, use the syntax `index [attrlist] [indices]`, where `[attrlist]` is a comma-separated list of the attributes and `[indices]` is a comma-separated list of the index types from Table 4 You may have several `index` lines. Specifying `default` as the attribute list sets the type of indexes that are used when the list of indexes is empty. Consider the indexes defined in Listing 20.

Listing 20. A sample set of indexes

```
index default eq,sub
index entryUUID,objectClass eq
index cn,sn,mail eq,sub,subinitial,subany,subfinal
index userid,telephonenumber
index ou eq
```

Listing 20 first defines the default indexes as an equality and a basic substring match. The second line places an equality index on the `entryUUID` attribute (good for syncrepl performance) and the `objectClass` attribute (a common search). Line 3 places an equality index and all substring indexes on the `cn`, `sn`, and `mail` attributes because these fields often have different wildcard searches. The `userid` and `telephonenumber` attributes receive the default indexes because nothing more specific was entered. Finally, the `ou` attribute has an equality index.

After changing your index definitions, you must rebuild the indexes by stopping slapd and running `slapindex` as the ldap user (or, if you're running as root, make sure to reassign ownership of all the files in the database directory to the ldap user after running `slapindex`). Start up slapd, and your indexes are used.

Section 6.

This section covers material for topic 303.6 for the Senior Level Linux Professional (LPIC-3) exam 301. This topic has a weight of 2.

In this section, you learn how to do the following:

- Understand slapd.conf configuration directives
- Understand slapd.conf database definitions
- Manage slapd and its command-line options
- Analyze slapd log files

The contents of slapd.conf have been largely covered earlier in this tutorial and in the previous tutorial. Of particular interest in this section are the command-line options and logging commands available to slapd.

Command-line options

The simplest way to start slapd is to run it without any arguments. Slapd then reads the default configuration file, forks to the background, and disassociates from the terminal.

Table 5 lists some helpful command-line arguments.

Table 5. slapd command-line arguments

Argument	Value	Description
`-d`	Integer	Runs slapd with extended debugging, and causes slapd to run in the foreground
`-f`	Filename	Specifies an alternate configuration file
`-h`	URL list	Specifies the addresses and ports that slapd listens on
`-s`	Sysloglevel	Specifies the syslog priority used for messages
`-l`	Integer	Specifies the local syslog facility (such as LOCAL4) used for messages
`-u`	Username	Runs slapd as the given user
`-g`	Groupname	Runs slapd in the given group

The URL list allows you to bind slapd to different interfaces. For example, `-h "ldap://127.0.0.1/ ldaps:///"` causes slapd to listen on TCP port 389 (unencrypted LDAP) only on the loopback, and encrypted LDAP (TCP port 636) on all interfaces. You can even alter port numbers: for example, `ldap://:5565/` causes unencrypted LDAP to run on port 5565 of all interfaces.

Understand logging

Slapd uses the Unix syslog daemon for logging. By default, all messages are sent to the `LOCAL4` facility, so you need at least `local4.* /var/log/openldap.log` in syslog.conf to capture the messages to /var/log/openldap.log. The `loglevel` command in slapd.conf then tells slapd what type of messages to log. Table 6 lists the possible message types.

Table 6. slapd message logging types

Keyword	Integer value	Description
`trace`	1	Trace function calls
`packet`	2	Debug packet handling
`args`	4	Heavy trace debugging

		(function args)
`conns`	8	Connection management
`BER`	16	Print out packets sent and received
`filter`	32	Search filter processing
`config`	64	Configuration file processing
`ACL`	128	ACL processing
`stats`	256	Stats log connections/operations/results
`stats2`	512	Stats log entries sent
`shell`	1024	Print communication with shell backends
`parse`	2048	Entry parsing
`sync`	16384	LDAPSync replication

You may use a space-separated list of keywords, integer values, or the sum of the integer values for the `loglevel` keyword. For example, `loglevel args ACL`, `loglevel 4 128`, and `loglevel 132` all enable debugging of function arguments and ACLs.

Section 7. Summary

In this tutorial, you learned about access control lists, replication, security, tuning, and more details about general configuration.

ACLs dictate who gets what access to what set of entries and attributes. You need to configure your ACLs using the form `access to <what> [ by <who> [ <access> ] [ <control> ] ]+`. You can use a variety of forms, including direct matches and regular expressions to specify the what. The who can also use matches and regular expressions, but it can also use keywords like `self`, `users`, and `anonymous`. The who clause can also look for things like the strength of the connection or the network the user is coming from.

Replication involves keeping a remote server up to date with the primary LDAP server. Two methods exist for replication: slurpd and syncrepl. In the slurpd model, a separate daemon runs on the master server and pushes all changes to the slaves. The slaves must start with a copy of the master's data; this requires downtime on the

master. In syncrepl, the provider (master) server runs an overlay to handle the replication tasks. The consumers (slaves) connect to the master and download any updates. If a consumer downloads updates on a periodic basis, it's said to be in refreshOnly mode. If the consumer downloads the updates and then stays connected, it's in refreshAndPersist mode, and it receives updates as they happen on the provider.

TLS and SSL let you encrypt communications between the client and server, and even replication traffic. You must generate server keys and have them signed by a CA for TLS to work. Regular LDAP traffic runs over TCP port 389, and encrypted LDAP traffic runs over TCP port 636, so you must have your firewalls configured accordingly.

Performance tuning involves assigning system resources to various caches and buffers and applying indexes on frequently searched columns. System resources are controlled in both the slapd.conf and DB_CONFIG files. Indexes can be equality, substring, presence, or approximate, depending on the type of search for which you're trying to optimize.

Most of slapd's behavior is controlled in slapd.conf, so there are only a few command-line parameters to control the addresses and ports that slapd listens on, the user it runs as, and some parameters about how it logs. What slapd logs is controlled by the `loglevel` directive in slapd.conf.

At this point, you have the skills to install, configure, and manage a functional OpenLDAP server, including security, replication, and performance tuning. The next two tutorials will focus on applications of LDAP, such as integrating LDAP with e-mail and authentication systems, and searching your tree from the command line.

Resources

Learn

- Review the previous tutorial in this 301 series, "LPI exam 301 prep, Topic 302: Installation and development" (developerWorks, December 2007), or all tutorials in the 301 series.
- Review the entire LPI exam prep tutorial series on developerWorks to learn Linux fundamentals and prepare for system administrator certification.
- At the LPIC Program, find task lists, sample questions, and detailed objectives for the three levels of the Linux Professional Institute's Linux system administration certification.
- Get the answer to How do I determine the proper BDB database cache size? in the OpenLDAP FAQ.
- Read about Backus-Naur form on Wikipedia for an in-depth understanding, including some examples. You may be familiar with BNF if you've looked into the LDAP Data Interchange Format (LDIF) or read some of the OpenLDAP manpages.
- The OpenLDAP Administrator's Guide has a chapter on ACLs that explains the syntax in detail. The `slapd.access(5)` manpage is a good companion to the Administrator's Guide.
- Also, see the chapters on syncrepl and slurpd replication in the OpenLDAP Administrator's Guide. In particular, the guide has detailed descriptions of how both replication types work.
- RFC 4533 (The Lightweight Directory Access Protocol (LDAP) Content Synchronization Operation), spearheaded by the OpenLDAP Foundation and IBM, describes a method to synchronize LDAP servers more effectively than can be done with `slurpd`.
- LDAP for Rocket Scientists, an online book, is excellent, despite being a work in progress.
- In the developerWorks Linux zone, find more resources for Linux developers, and scan our most popular articles and tutorials.
- See all Linux tips and Linux tutorials on developerWorks.
- Stay current with developerWorks technical events and Webcasts.

Get products and technologies

- The Firewall Builder utility makes the task of typing in iptables rules easy; it has a nice GUI and suite of tools to roll out updates to your firewalls.

- OpenLDAP is a great choice for an LDAP server.
- phpLDAPadmin is a Web-based LDAP administration tool. If the GUI is more your style, Luma is a good one to look at.
- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- With IBM trial software, available for download directly from developerWorks, build your next development project on Linux.

Discuss

- Get involved in the developerWorks community through blogs, forums, podcasts, and community topics in our new developerWorks spaces.

About the author

Sean A. Walberg
Sean Walberg has been working with Linux and UNIX since 1994 in academic, corporate, and Internet service provider environments. He has written extensively about systems administration over the past several years.

Trademarks